DEPARTMENT OF MATHEMATICS AND COMPUTING
V-M.Tech. (M&C)
Monsoon Semester 2022-2023

GPU Computing Lab
MCC302

LAB-2

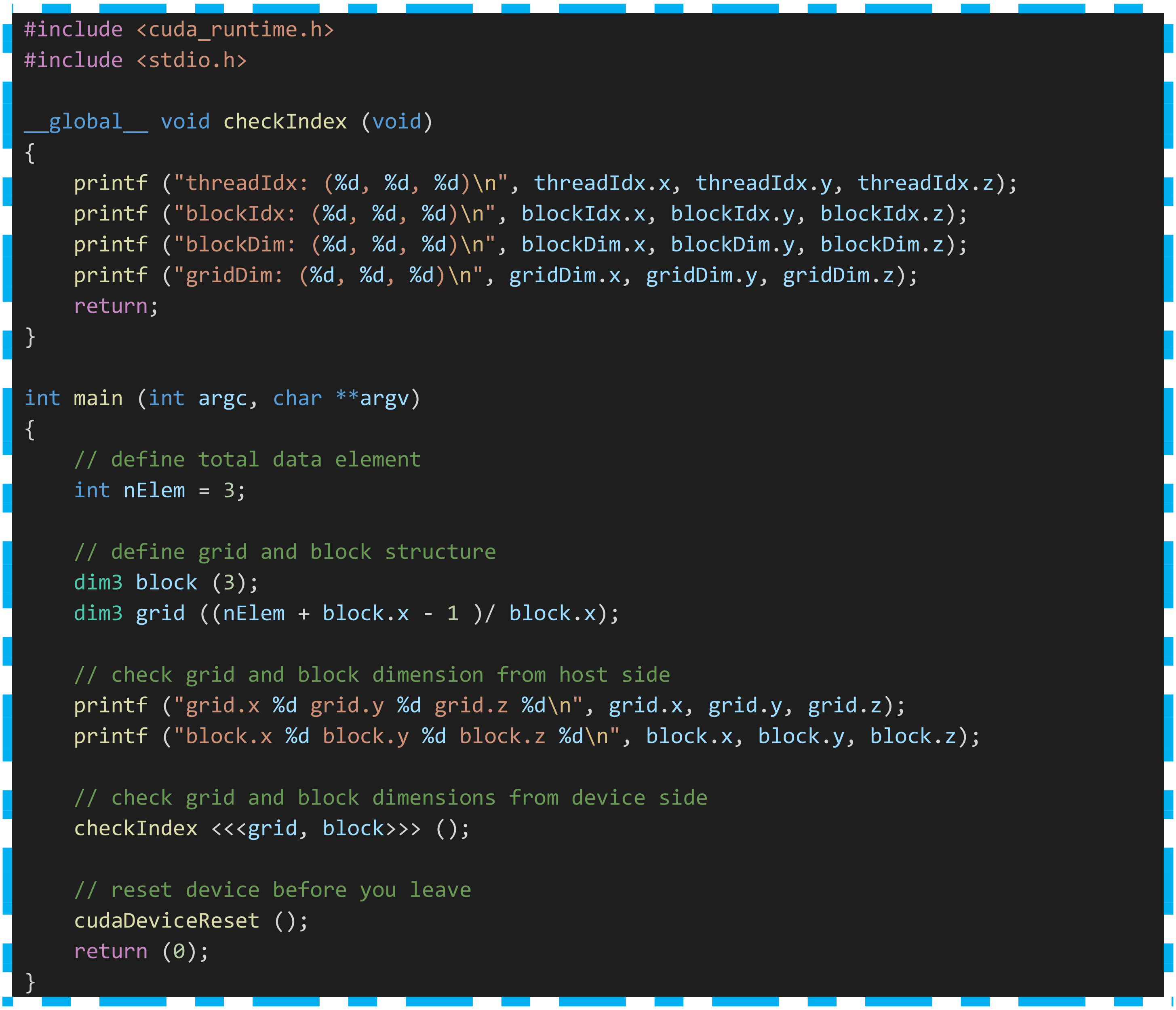
Vector Sum and Dot Product

NAME: **ROMEO SARKAR**
ADMISSION NO.: **20JE0814**
DATE: **17-08-2022**

Experiment 2.1: Display the dimensions of grid and a thread block.

Objectives: Display the number of threads in block and number of blocks in the grid.

CUDA Sample Program:



```
#include <cuda_runtime.h>
#include <stdio.h>

__global__ void checkIndex (void)
{
    printf ("threadIdx: (%d, %d, %d)\n", threadIdx.x, threadIdx.y, threadIdx.z);
    printf ("blockIdx: (%d, %d, %d)\n", blockIdx.x, blockIdx.y, blockIdx.z);
    printf ("blockDim: (%d, %d, %d)\n", blockDim.x, blockDim.y, blockDim.z);
    printf ("gridDim: (%d, %d, %d)\n", gridDim.x, gridDim.y, gridDim.z);
    return;
}

int main (int argc, char **argv)
{
    // define total data element
    int nElem = 3;

    // define grid and block structure
    dim3 block (3);
    dim3 grid ((nElem + block.x - 1 )/ block.x);

    // check grid and block dimension from host side
    printf ("grid.x %d grid.y %d grid.z %d\n", grid.x, grid.y, grid.z);
    printf ("block.x %d block.y %d block.z %d\n", block.x, block.y, block.z);

    // check grid and block dimensions from device side
    checkIndex <<<grid, block>>> ();

    // reset device before you leave
    cudaDeviceReset ();
    return (0);
}
```

Output:

```
grid.x 1 grid.y 1 grid.z 1
block.x 3 block.y 1 block.z 1
threadIdx: (0, 0, 0)
threadIdx: (1, 0, 0)
threadIdx: (2, 0, 0)
blockIdx: (0, 0, 0)
blockIdx: (0, 0, 0)
blockIdx: (0, 0, 0)
blockDim: (3, 1, 1)
blockDim: (3, 1, 1)
blockDim: (3, 1, 1)
gridDim: (1, 1, 1)
gridDim: (1, 1, 1)
gridDim: (1, 1, 1)
```

Experiment 2.2: Define grid and Blocks.

Objectives: Display grid and block structure.

CUDA Sample Program:

```
#include <cuda_runtime.h>
#include <stdio.h>
int main (int argc, char **argv)
{
    // define total data element
    int nElem = 1024;

    // define grid and block size
    dim3 block (1024);
    dim3 grid ((nElem + block.x - 1) / block.x);
    printf ("grid.x %d block.x %d\n", grid.x, block.x);

    // reset block
    block.x = 512;
    grid.x = (nElem + block.x - 1) / block.x;
    printf ("grid.x %d block.x %d\n", grid.x, block.x);

    // reset block
    block.x = 256;
    grid.x = (nElem + block.x - 1) / block.x;
    printf ("grid.x %d block.x %d\n", grid.x, block.x);

    // reset block
    block.x = 128;
    grid.x = (nElem + block.x - 1) / block.x;
    printf ("grid.x %d block.x %d\n", grid.x, block.x);

    // reset device before you leave
    cudaDeviceReset ();

    return 0;
}
```

Output:

```
grid.x 1 block.x 1024
grid.x 2 block.x 512
grid.x 4 block.x 256
grid.x 8 block.x 128
```

Experiment 2.3: Vector Addition on GPU.

Objectives: Element wise sum of vector.

CUDA Sample Program:

```
#include <cuda_runtime.h>
#include <stdio.h>
#define N 10
__global__ void VecAddGPU (int *a, int *b, int *c)
{
    int i = blockIdx.x;
    if (i < N)
    {
        c[i] = a[i] + b[i];
    }
    return;
}

int main (int argc, char **argv)
{
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;
    // allocate memory on device
    cudaMalloc ((void **) (&dev_a), N * sizeof (int));
    cudaMalloc ((void **) (&dev_b), N * sizeof (int));
    cudaMalloc ((void **) (&dev_c), N * sizeof (int));
    for (int i = 0; i < N; i++)
    {
        a[i] = -i;
        b[i] = i * i;
    }

    // Copy data from host to device
    cudaMemcpy (dev_a, a, N * sizeof (int), cudaMemcpyHostToDevice);
    cudaMemcpy (dev_b, b, N * sizeof (int), cudaMemcpyHostToDevice);

    // launch kernel
    VecAddGPU <<<N, 1>>> (dev_a, dev_b, dev_c);
    // Copy result from device to host
    cudaMemcpy (c, dev_c, N * sizeof (int), cudaMemcpyDeviceToHost);

    for (int i = 0; i < N; i++)
    {
        printf ("%d + %d = %d\n", a[i], b[i], c[i]);
    }
    cudaFree (dev_a);
    cudaFree (dev_b);
    cudaFree (dev_c);
    return 0;
}
```

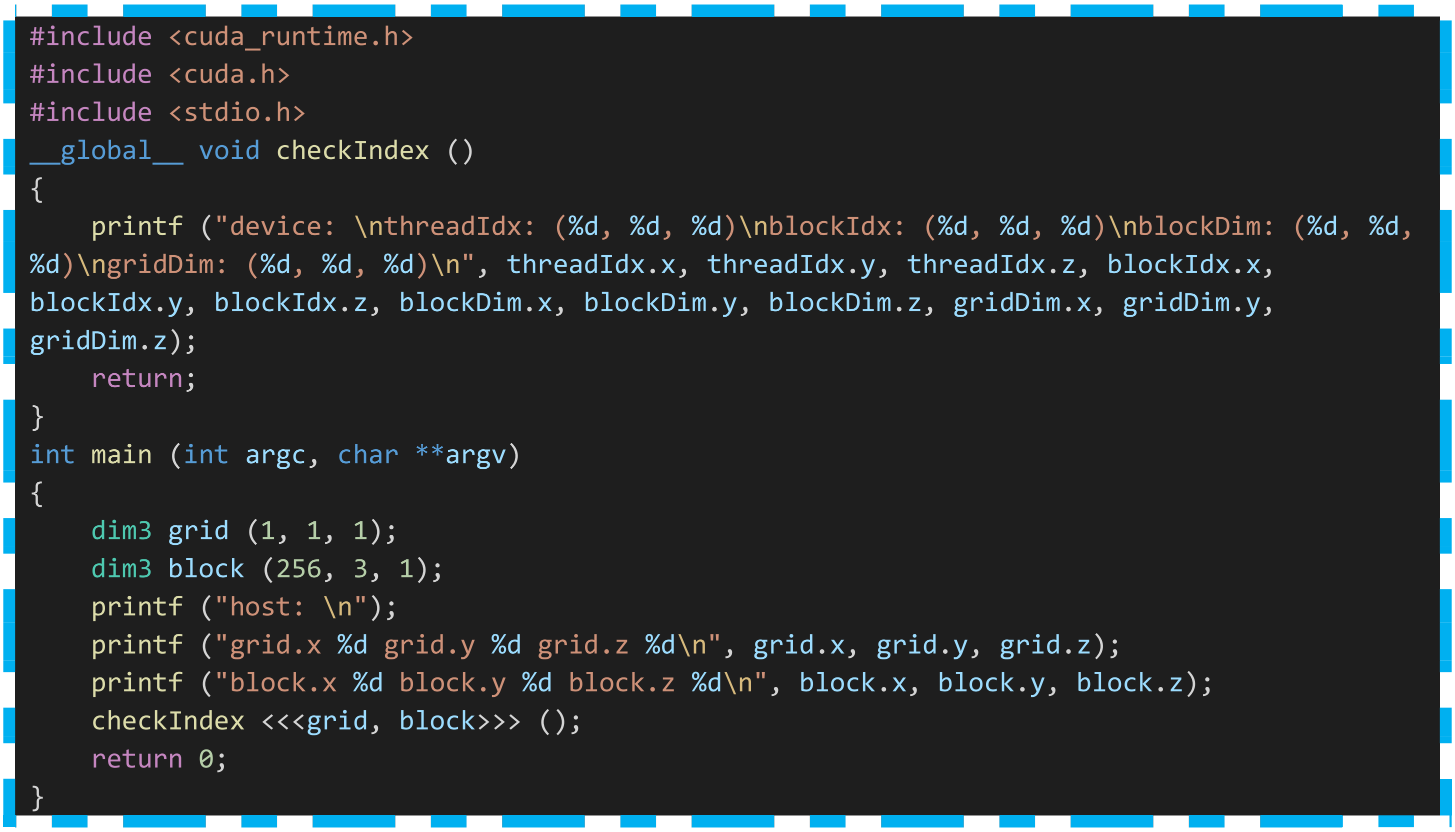
Output:

```
0 + 0 = 0
-1 + 1 = 0
-2 + 4 = 2
-3 + 9 = 6
-4 + 16 = 12
-5 + 25 = 20
-6 + 36 = 30
-7 + 49 = 42
-8 + 64 = 56
-9 + 81 = 72
```

Lab Exercise 2.1: Write a CUDA program to display:

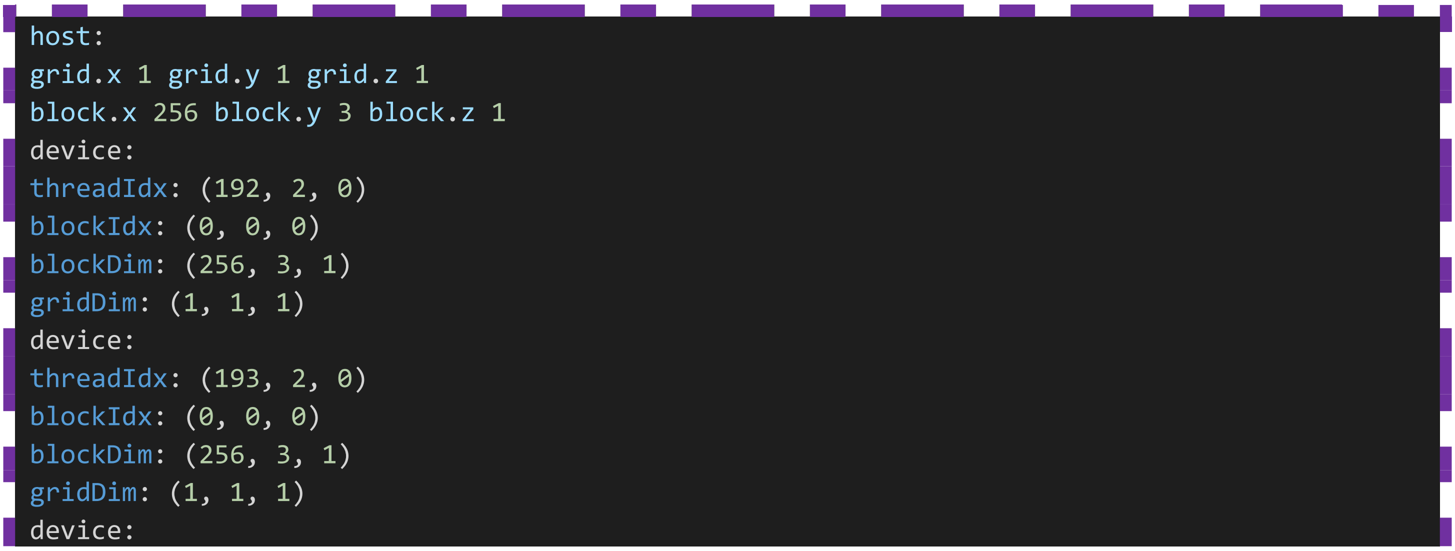
1) Display grid, block and thread details for a block of size (256, 3, 1).

CODE1:



```
#include <cuda_runtime.h>
#include <cuda.h>
#include <stdio.h>
__global__ void checkIndex ()
{
    printf ("device: \nthreadIdx: (%d, %d, %d)\nblockIdx: (%d, %d, %d)\nblockDim: (%d, %d,
%d)\ngridDim: (%d, %d, %d)\n", threadIdx.x, threadIdx.y, threadIdx.z, blockIdx.x,
blockIdx.y, blockIdx.z, blockDim.x, blockDim.y, blockDim.z, gridDim.x, gridDim.y,
gridDim.z);
    return;
}
int main (int argc, char **argv)
{
    dim3 grid (1, 1, 1);
    dim3 block (256, 3, 1);
    printf ("host: \n");
    printf ("grid.x %d grid.y %d grid.z %d\n", grid.x, grid.y, grid.z);
    printf ("block.x %d block.y %d block.z %d\n", block.x, block.y, block.z);
    checkIndex <<<grid, block>>> ();
    return 0;
}
```

Output1:



```
host:
grid.x 1 grid.y 1 grid.z 1
block.x 256 block.y 3 block.z 1
device:
threadIdx: (192, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (193, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
```

```
threadIdx: (194, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (195, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (196, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (197, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (198, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (199, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (200, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (201, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
device:
threadIdx: (202, 2, 0)
blockIdx: (0, 0, 0)
blockDim: (256, 3, 1)
gridDim: (1, 1, 1)
.
.
.
```

Lab Exercise 2.2: Write a CUDA program to display:

1) Distance between two vectors x and y where x = $\{i^2\}_{i=1}^{n}$, y = $\{(2i+1)\}_{i=1}^{n}$ and n = 1024. Also find the Euclidean norms of x and y respectively.
2) Find the standard deviation of y = $\{(2i+1)\}_{i=1}^{n}$ and n = 1024.

CODE1:

```
#include <cuda_runtime.h>
#include <stdio.h>
#define N 1024
__global__ void calcSqOfDiff (double *a, double *b, double *c)
{
    int i = blockIdx.x;
    if (i < N)
    {
        c[i] = a[i] - b[i];
        c[i] *= c[i];
    }
    return;
}

__global__ void calcSq (double *a, double *b)
{
    int i = blockIdx.x;
    if (i < N)
    {
        b[i] = a[i] * a[i];
    }
    return;
}

int main (int argc, char **argv)
{
    double x[N], y[N], z[N];
    double *dev_x, *dev_y, *dev_z;
    // allocate memory on device
    cudaMalloc ((void **) (&dev_x), N * sizeof (double));
    cudaMalloc ((void **) (&dev_y), N * sizeof (double));
    cudaMalloc ((void **) (&dev_z), N * sizeof (double));
    for (int c = 0, i = 1; c < N; c++, i++)
    {
        x[c] = i * i;
        y[c] = 2 * i + 1;
    }

    // Copy data from host to device
    cudaMemcpy (dev_x, x, N * sizeof (double), cudaMemcpyHostToDevice);
    cudaMemcpy (dev_y, y, N * sizeof (double), cudaMemcpyHostToDevice);

    // launch kernel
    calcSqOfDiff <<<N, 1>>> (dev_x, dev_y, dev_z);
    cudaDeviceSynchronize ();
    // wait for kernel to return
    // Copy result from device to host
    cudaMemcpy (z, dev_z, N * sizeof (double), cudaMemcpyDeviceToHost);
    double sumOfSq = 0;
    for (int c = 0; c < N; c++)
    {
        // printf ("%lf\n", z[c]);
        sumOfSq += z[c];
    }
    // printf ("sum of squares: %lf\n", sumOfSq);
    printf ("Distance between x and y is %lf\n", sqrt (sumOfSq));
    // calculating norm of x
    calcSq <<<N, 1>>> (dev_x, dev_z);
    cudaDeviceSynchronize ();
    cudaMemcpy (z, dev_z, N * sizeof (double), cudaMemcpyDeviceToHost);
    double sumOfSq_x = 0;
    for (int c = 0; c < N; c++)
    {
        // printf ("%lf\n", z[c]);
        sumOfSq_x += z[c];
    }
    printf ("Norm of x: %lf\n", sqrt (sumOfSq_x));
    // calculating norm of y
    calcSq <<<N, 1>>> (dev_y, dev_z);
    cudaDeviceSynchronize ();
    cudaMemcpy (z, dev_z, N * sizeof (double), cudaMemcpyDeviceToHost);
    double sumOfSq_y = 0;
    for (int c = 0; c < N; c++)
    {
        // printf ("%lf\n", z[c]);
        sumOfSq_y += z[c];
    }
    printf ("Norm of y: %lf\n", sqrt (sumOfSq_y));
    cudaFree (dev_x);
    cudaFree (dev_y);
    cudaFree (dev_z);
    cudaDeviceReset ();
    return 0;
}
```

Output1:

```
Distance between x and y is 14987633.365990
Norm of x: 15024316.792997
Norm of y: 37892.661875
```

CODE2:

```
#include <cuda_runtime.h>
#include <stdio.h>
#define N 1024

__global__ void findSqOfDiff (double *a, double *b, double mean)
{
    int i = blockIdx.x;
    if (i < N)
    {
        b[i] = mean - a[i];
        b[i] *= b[i];
    }
    return;
}

int main (int argc, char **argv)
{
    double arr[N], sqOfDiffFromMean[N];
    double *dev_a1, *dev_a2;
    cudaMalloc (&dev_a1, N * sizeof (double));
    cudaMalloc (&dev_a2, N * sizeof (double));
    // cudaMalloc ()
    double mean = 0;
    for (int c = 0, i = 1; c < N; c++, i++)
    {
        arr[c] = (2 * i + 1);
        mean += arr[c];
    }
    mean /= N;
    printf ("Mean: %lf\n", mean);
    cudaMemcpy (dev_a1, arr, N * sizeof (double), cudaMemcpyHostToDevice);

    findSqOfDiff <<<N, 1>>> (dev_a1, dev_a2, mean);
    cudaDeviceSynchronize ();
    cudaMemcpy (sqOfDiffFromMean, dev_a2, N * sizeof (double), cudaMemcpyDeviceToHost);
    double s = 0;
    for (int i = 0; i < N; i++)
    {
        // printf ("%lf\n", sqOfDiffFromMean[i]);
        s += sqOfDiffFromMean[i];
    }
    s /= N;
    printf ("Standard Deviation: %lf\n", sqrt (s));
    cudaFree (dev_a1);
    cudaFree (dev_a2);
    cudaDeviceReset ();
    return 0;
}
```

Output2:

```
Mean: 1026.000000
Standard Deviation: 591.206394
```